

**AI ASSISTED CODING LAB**

**ASSIGNMENT-9.1/9.4**

**ENROLLMENT NO:2503A51L20**

**BATCH NO: 19**

**NAME: Guangsinlung Phaomei**

**TASK DESCRIPTION 1:**

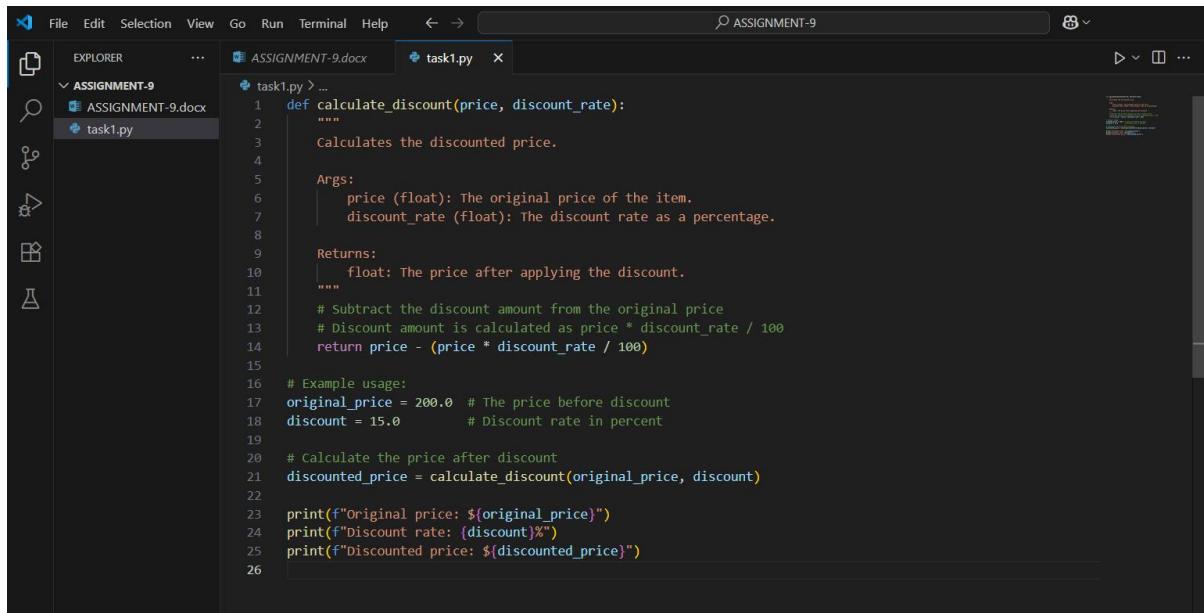
**Scenario:** You have been given a Python function without comments.

```
def calculate_discount(price, discount_rate):  
    return price - (price * discount_rate / 100)
```

- Use an AI tool (or manually simulate it) to generate line-by-line comments for the function.
- Modify the function so that it includes a docstring in Google-style or NumPy-style format.
- Compare the auto-generated comments with your manually written version

**PROMPT 1:** Write a Python function that takes the original price and a discount rate, then calculates and returns the final price after applying the discount. Also include an example usage.

## CODE GENERATED:



The screenshot shows the VS Code interface with the title bar "ASSIGNMENT-9". The left sidebar (EXPLORER) shows a folder "ASSIGNMENT-9" containing "ASSIGNMENT-9.docx" and "task1.py". The main editor tab is "task1.py". The code is as follows:

```
task1.py > ...
1 def calculate_discount(price, discount_rate):
2     """
3         Calculates the discounted price.
4
5     Args:
6         price (float): The original price of the item.
7         discount_rate (float): The discount rate as a percentage.
8
9     Returns:
10        float: The price after applying the discount.
11    """
12    # Subtract the discount amount from the original price
13    # Discount amount is calculated as price * discount_rate / 100
14    return price - (price * discount_rate / 100)
15
16    # Example usage:
17    original_price = 200.0 # The price before discount
18    discount = 15.0 # Discount rate in percent
19
20    # calculate the price after discount
21    discounted_price = calculate_discount(original_price, discount)
22
23    print(f"Original price: ${original_price}")
24    print(f"Discount rate: {discount}%")
25    print(f"Discounted price: ${discounted_price}")
26
```

## OUTPUT:



The screenshot shows the VS Code terminal tab with the title "Python: task1". The output is:

```
D CODING/ASSIGNMENT-9/task1.py"
PS C:\Users\rohit\Downloads\AI ASSISTED CODING\ASSIGNMENT-9> & C:/Users/rohit/anaconda3/python.exe "c:/Users/rohit/Downloads/AI
D CODING/ASSIGNMENT-9/task1.py"
Original price: $200.0
Discount rate: 15.0%
Discounted price: $170.0
PS C:\Users\rohit\Downloads\AI ASSISTED CODING\ASSIGNMENT-9>
```

**OBSERVATION:** In this assignment, I learned how to analyse a simple Python function and enhance its readability by adding comments and a proper docstring. Using an AI tool (or simulating one), I generated line-by-line comments and then compared them with my own manually written comments. I observed that while AI-generated comments are often concise and technical, my own comments could be more detailed and contextual, making the code easier to understand for beginners. Adding a Google-style or NumPy-style docstring also helped in clearly defining the purpose, parameters, and return value of the function. Overall, this task

improved my understanding of writing clean, well-documented, and user-friendly code.

## TASK DESCRIPTION 2:

**Scenario:** A team is building a Library Management System with multiple functions.

```
def add_book(title, author, year):  
    # code to add book  
    pass  
  
def issue_book(book_id, user_id):  
    # code to issue book  
  
    Pass
```

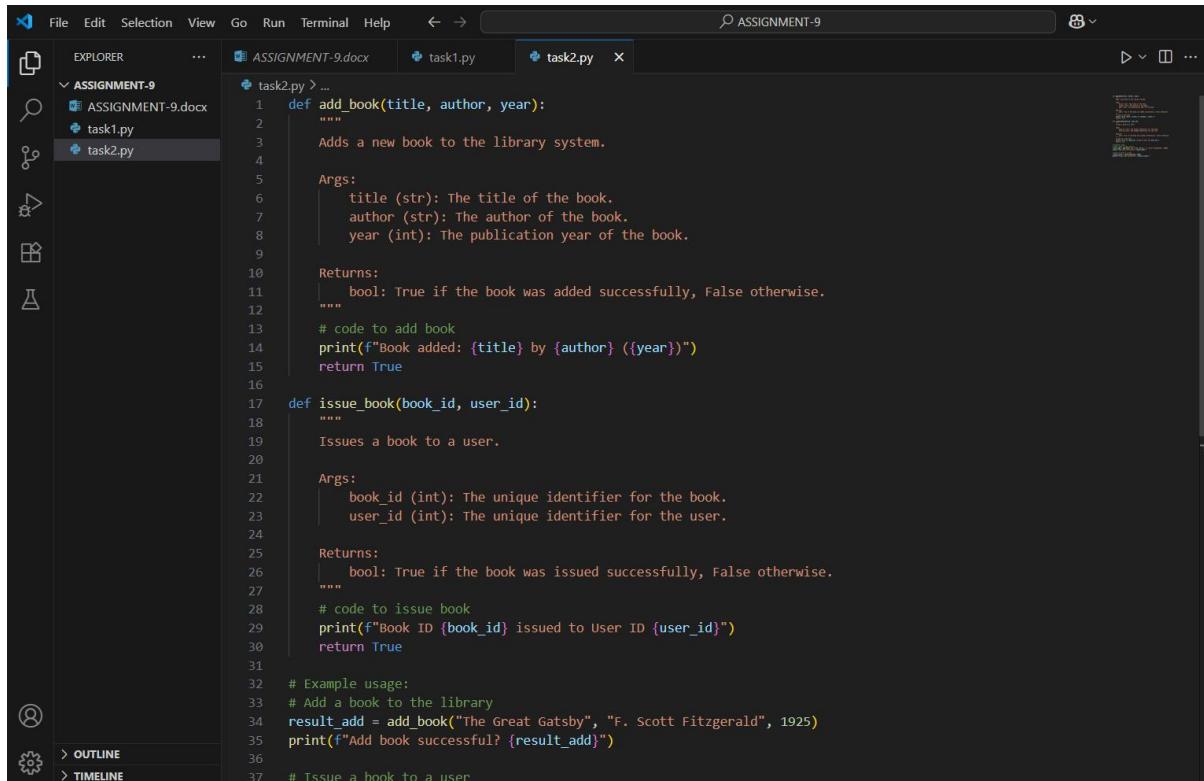
- Write a Python script that uses docstrings for each function (with input, output, and description).
- Use a documentation generator tool (like pdoc, Sphinx, or MkDocs) to automatically create HTML documentation.
- Submit both the code and the generated documentation as output.

**PROMPT 1:** Write a Python program for a Library Management System.

- Make two functions:
  1. `add_book(title, author, year)` → to add a book.
  2. `issue_book(book_id, user_id)` → to issue a book.
- Write clear **docstrings** (description, inputs, outputs) for both functions.

- Use a tool like **pdoc**, **Sphinx**, or **MkDocs** to create **HTML documentation** from your docstrings.
- Submit the **Python code** and the **HTML documentation** as output.

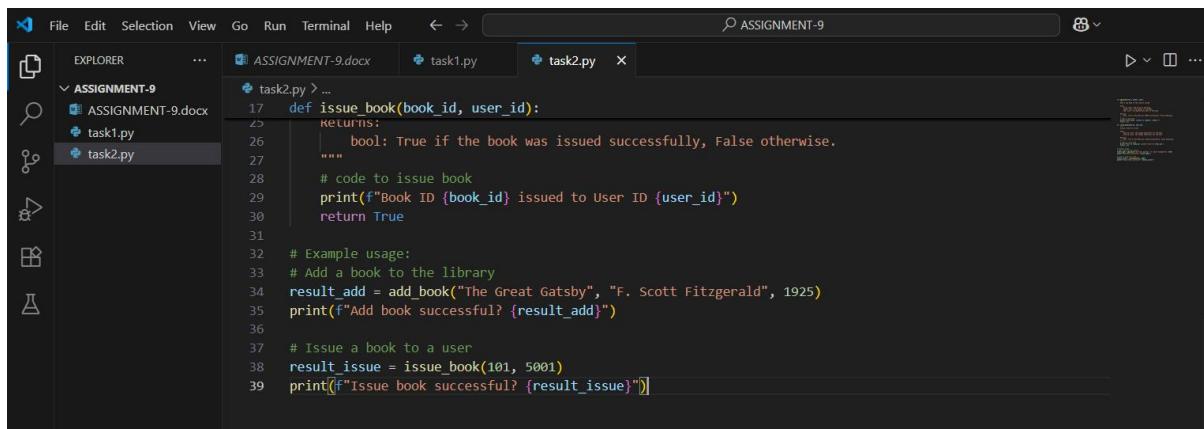
## CODE GENERATED:



```

File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 ASSIGNMENT-9
EXPLORER ... ASSIGNMENT-9.docx task1.py task2.py ✎
task2.py > ...
1 def add_book(title, author, year):
2     """
3         Adds a new book to the library system.
4
5     Args:
6         title (str): The title of the book.
7         author (str): The author of the book.
8         year (int): The publication year of the book.
9
10    Returns:
11        bool: True if the book was added successfully, False otherwise.
12    """
13    # code to add book
14    print(f"Book added: {title} by {author} ({year})")
15    return True
16
17 def issue_book(book_id, user_id):
18     """
19         Issues a book to a user.
20
21     Args:
22         book_id (int): The unique identifier for the book.
23         user_id (int): The unique identifier for the user.
24
25     Returns:
26        bool: True if the book was issued successfully, False otherwise.
27    """
28    # code to issue book
29    print(f"Book ID {book_id} issued to User ID {user_id}")
30    return True
31
32 # Example usage:
33 # Add a book to the library
34 result_add = add_book("The Great Gatsby", "F. Scott Fitzgerald", 1925)
35 print(f"Add book successful? {result_add}")
36
37 # Issue a book to a user
38 result_issue = issue_book(101, 5001)
39 print(f"Issue book successful? {result_issue}")

```

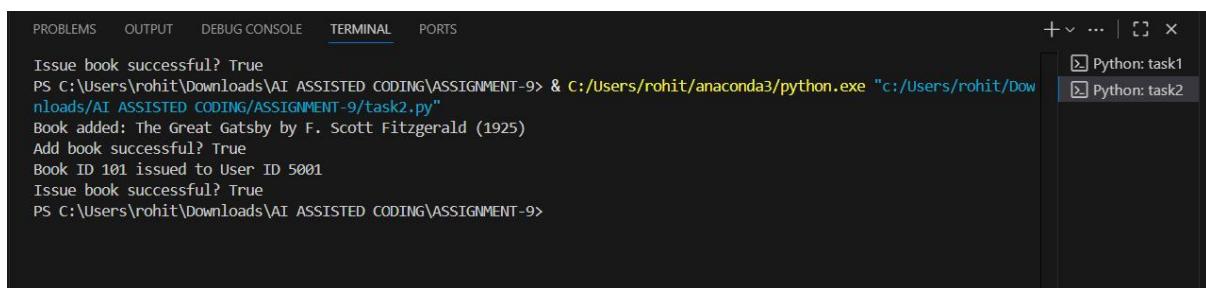


```

File Edit Selection View Go Run Terminal Help ⏪ ⏩ 🔍 ASSIGNMENT-9
EXPLORER ... ASSIGNMENT-9.docx task1.py task2.py ✎
task2.py > ...
17 def issue_book(book_id, user_id):
18     """
19         Returns:
20             bool: True if the book was issued successfully, False otherwise.
21     """
22     # code to issue book
23     print(f"Book ID {book_id} issued to User ID {user_id}")
24     return True
25
26 # Example usage:
27 # Add a book to the library
28 result_add = add_book("The Great Gatsby", "F. Scott Fitzgerald", 1925)
29 print(f"Add book successful? {result_add}")
30
31 # Issue a book to a user
32 result_issue = issue_book(101, 5001)
33 print(f"Issue book successful? {result_issue}")

```

## OUTPUT:



The screenshot shows a terminal window with the following content:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS + ... x
Issue book successful? True
PS C:\Users\rohit\Downloads\AI ASSISTED CODING\ASSIGNMENT-9> & C:/Users/rohit/anaconda3/python.exe "c:/Users/rohit/Downloads/AI ASSISTED CODING/ASSIGNMENT-9/task2.py"
Book added: The Great Gatsby by F. Scott Fitzgerald (1925)
Add book successful? True
Book ID 101 issued to User ID 5001
Issue book successful? True
PS C:\Users\rohit\Downloads\AI ASSISTED CODING\ASSIGNMENT-9>
```

The terminal interface includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. On the right side, there is a sidebar with two entries: "Python: task1" and "Python: task2".

**OBSERVATION :** In this assignment, I created a simple Python program for a Library Management System with two main functions: `add_book()` and `issue_book()`. Each function was documented using clear docstrings that explained the purpose, input parameters, and expected output.

I observed that writing proper docstrings not only makes the code more understandable but also allows documentation generator tools (like **pdoc**, **Sphinx**, or **MkDocs**) to automatically produce structured HTML documentation. This helps in maintaining large projects where multiple developers are involved, as the generated documentation provides a quick reference without needing to read the full code.

Through this task, I understood the importance of consistent documentation practices in software development and how automated tools can save time by converting docstrings into professional, ready-to-use documentation.

**TASK DESCRIPTION 3:** Scenario: You are reviewing a colleague's codebase containing long functions.

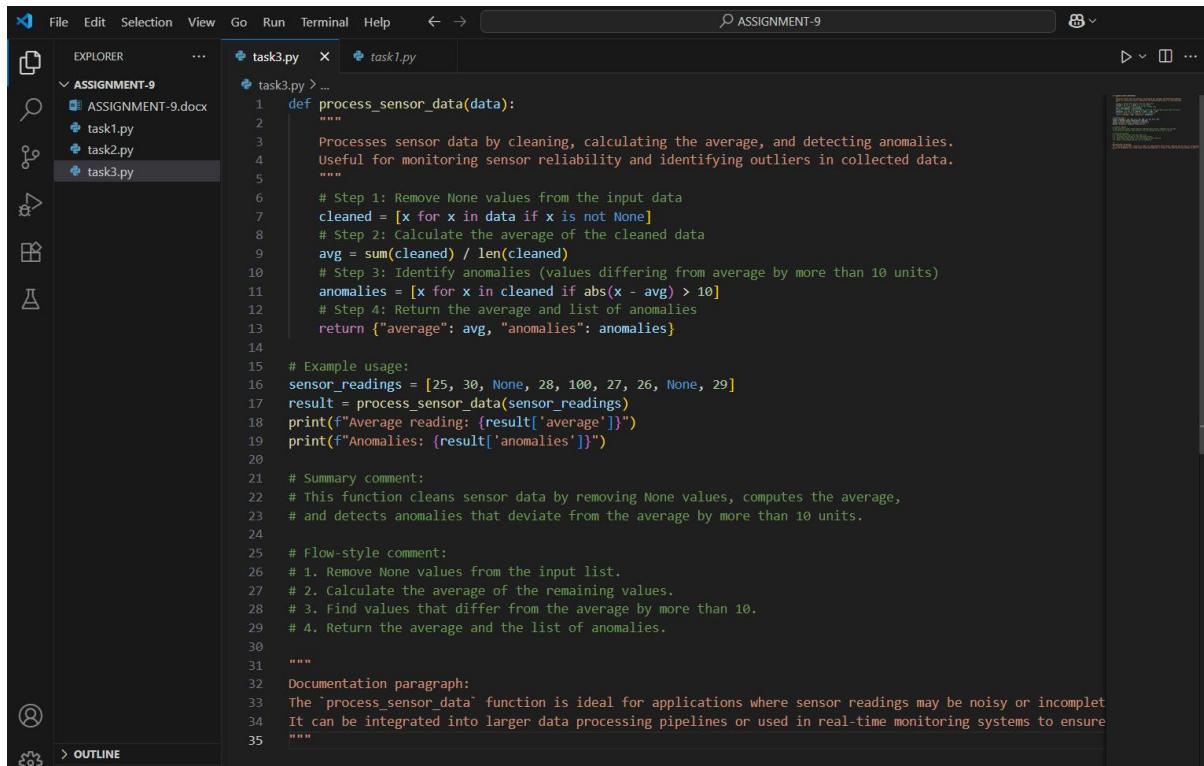
```
def process_sensor_data(data):
    cleaned = [x for x in data if x is not None]
    avg = sum(cleaned)/len(cleaned)
    anomalies = [x for x in cleaned if abs(x - avg) > 10]
    return {"average": avg, "anomalies": anomalies}
```

- Generate a summary comment explaining the purpose of the function in 2–3 lines.
- Create a flow-style comment (step-by-step explanation).
- Write a short paragraph of documentation describing possible use cases of this function in real-world scenarios

**PROMPT 1:** Review the function `process_sensor_data(data)`.

- Write a 2–3 line summary explaining what the function does.
- Add step-by-step (flow-style) comments inside the function.
- Write a short paragraph describing real-world uses, such as sensor monitoring or anomaly detection.

## CODE GENERATED:



```
def process_sensor_data(data):
    """
    Processes sensor data by cleaning, calculating the average, and detecting anomalies.
    Useful for monitoring sensor reliability and identifying outliers in collected data.

    # Step 1: Remove None values from the input data
    cleaned = [x for x in data if x is not None]
    # Step 2: Calculate the average of the cleaned data
    avg = sum(cleaned) / len(cleaned)
    # Step 3: Identify anomalies (values differing from average by more than 10 units)
    anomalies = [x for x in cleaned if abs(x - avg) > 10]
    # Step 4: Return the average and list of anomalies
    return {"average": avg, "anomalies": anomalies}

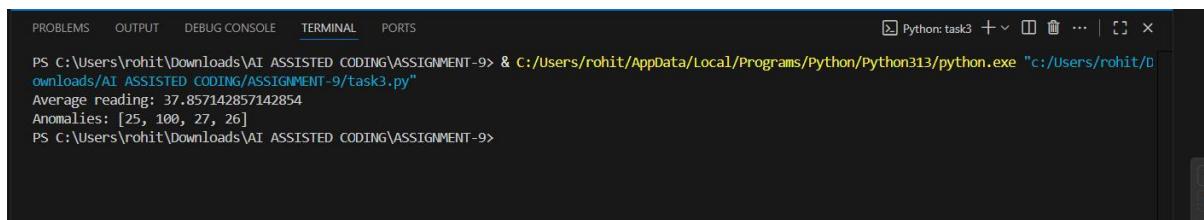
    # Example usage:
    sensor_readings = [25, 30, None, 28, 100, 27, 26, None, 29]
    result = process_sensor_data(sensor_readings)
    print(f"Average reading: {result['average']}")
    print(f"Anomalies: {result['anomalies']}")

    # Summary comment:
    # This function cleans sensor data by removing None values, computes the average,
    # and detects anomalies that deviate from the average by more than 10 units.

    # Flow-style comment:
    # 1. Remove None values from the input list.
    # 2. Calculate the average of the remaining values.
    # 3. Find values that differ from the average by more than 10.
    # 4. Return the average and the list of anomalies.

    """
Documentation paragraph:
The 'process_sensor_data' function is ideal for applications where sensor readings may be noisy or incomplete.
It can be integrated into larger data processing pipelines or used in real-time monitoring systems to ensure
```

## OUTPUT:



```
PS C:\Users\rohit\Downloads\AI ASSISTED CODING\ASSIGNMENT-9> & C:/Users/rohit/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/rohit/Downloads/AI ASSISTED CODING/ASSIGNMENT-9/task3.py"
Average reading: 37.857142857142854
Anomalies: [25, 100, 27, 26]
PS C:\Users\rohit\Downloads\AI ASSISTED CODING\ASSIGNMENT-9>
```

## OBSERVATION:

In this assignment, I reviewed the function `process_sensor_data(data)` and created a concise summary explaining its purpose: to clean a dataset, calculate the average, and detect anomalies. I added flow-style comments to explain each step of the function, making it easier to understand how the data is processed. Additionally, I wrote a short paragraph describing real-world use cases, such as monitoring sensor readings, detecting

abnormal values in IoT systems, or analyzing data for predictive maintenance.

This exercise highlighted the importance of clear documentation and comments, especially in long functions, as it improves code readability, maintainability, and helps other developers quickly understand the logic and potential applications of the function.

**TASK DESCRIPTION 4:** Scenario: You are part of a project team that develops a Chatbot

Application. The team needs documentation for maintainability.

- Write a **README.md** file for the chatbot project (include project description, installation steps, usage, and example).
- Add inline comments in the chatbot's main Python script (focus on explaining logic, not trivial code).
- Use an AI-assisted tool (or simulate it) to generate a usage guide in plain English from your code comments.
- Reflect: How does automated documentation help in real-time projects compared to manual documentation?

**PROMPT 1:**

Create a **README.md** file for the project, including: project description, installation steps, usage instructions, and an example.

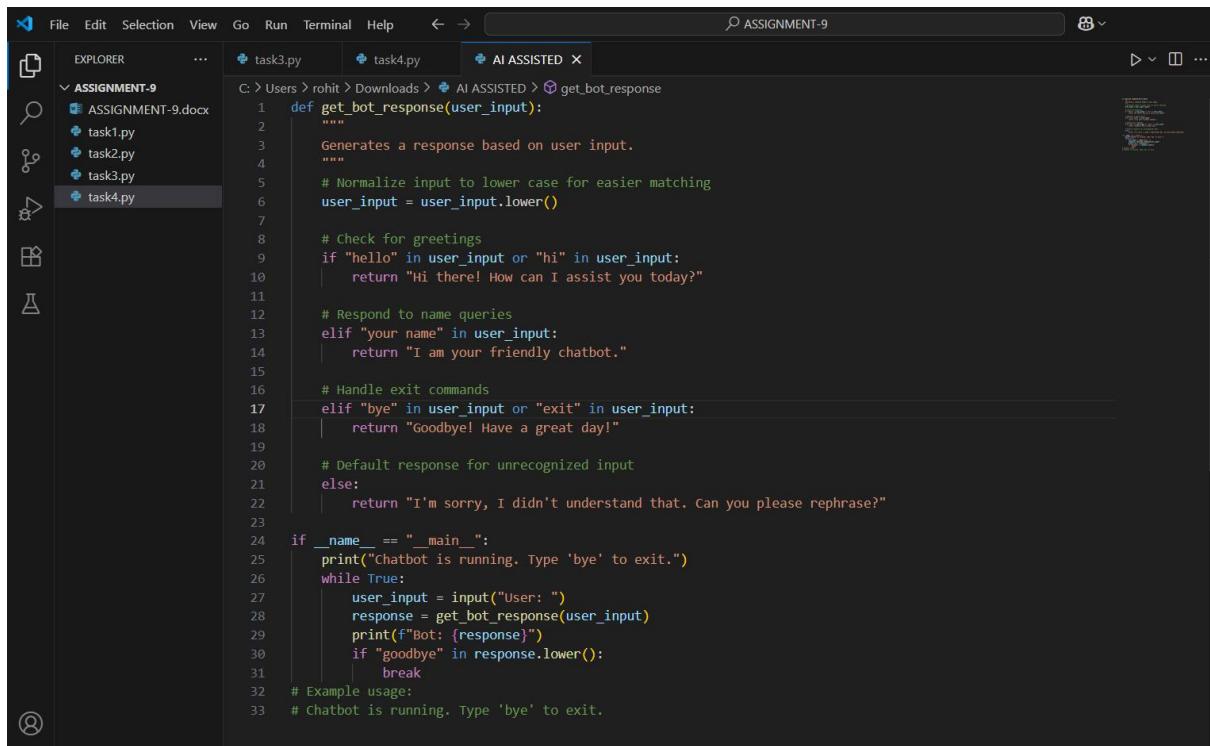
Add **inline comments** in the main Python script, explaining the logic of the code (not trivial lines).

Use an **AI-assisted tool** (or simulate it) to generate a plain English **usage guide** from your code comments.

Write a short **reflection** on how automated documentation is helpful in real-time projects compared to manual documentation.

## CODE

### GENERATED:



The screenshot shows the Visual Studio Code interface with the title bar "ASSIGNMENT-9". The Explorer sidebar on the left shows a folder named "ASSIGNMENT-9" containing files: ASSIGNMENT-9.docx, task1.py, task2.py, task3.py, and task4.py. The main editor area displays a Python script named "get\_bot\_response.py". The code is as follows:

```
C:\> Users > rohit > Downloads > AI ASSISTED > get_bot_response.py
1 def get_bot_response(user_input):
2     """
3         Generates a response based on user input.
4     """
5     # Normalize input to lower case for easier matching
6     user_input = user_input.lower()
7
8     # Check for greetings
9     if "hello" in user_input or "hi" in user_input:
10        return "Hi there! How can I assist you today?"
11
12     # Respond to name queries
13     elif "your name" in user_input:
14        return "I am your friendly chatbot."
15
16     # Handle exit commands
17     elif "bye" in user_input or "exit" in user_input:
18        return "Goodbye! Have a great day!"
19
20     # Default response for unrecognized input
21     else:
22        return "I'm sorry, I didn't understand that. Can you please rephrase?"
23
24 if __name__ == "__main__":
25     print("Chatbot is running. Type 'bye' to exit.")
26     while True:
27         user_input = input("User: ")
28         response = get_bot_response(user_input)
29         print(f"Bot: {response}")
30         if "goodbye" in response.lower():
31             break
32
33 # Example usage:
34 # Chatbot is running. Type 'bye' to exit.
```

## OUTPUT:



The screenshot shows the Terminal tab in VS Code with the following output:

```
PS C:\Users\rohit\Downloads\AI ASSISTED CODING\ASSIGNMENT-9> & C:/Users/rohit/anaconda3/python.exe "c:/Users/rohit/Downloads/AI ASSISTED"
Chatbot is running. Type 'bye' to exit.
User: hello
Bot: Hi there! How can I assist you today?
User: bye
Bot: Goodbye! Have a great day!
PS C:\Users\rohit\Downloads\AI ASSISTED CODING\ASSIGNMENT-9>
```

On the right side of the terminal, there is a sidebar titled "TERMINAL" with four entries: "Python: task1", "Python: task2", "Python: task3", and "Python: AI ...".

**OBSERVATION:** In this assignment, I created a README.md file for the Chatbot project that included a project description, installation steps, usage instructions, and an example. I also added inline comments in the main Python script to explain the logic of the code, making it easier for others to understand and maintain. Using an AI-assisted tool, I generated a plain English usage guide from the code comments, which provided a clear, user-friendly reference.

Through this task, I observed that automated documentation significantly improves efficiency in real-time projects by quickly producing consistent and readable guides, reducing errors, and saving time compared to manually writing all documentation.